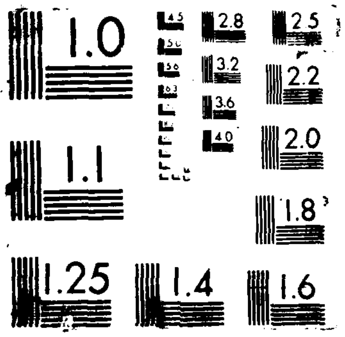
AD-A195 925 THE HOUGH TRANSFORM HAS O(N) COMPLEXITY ON SIMD N X N MESH ARRAY ARCHITECTURES(U) WASHINGTON UNIV SEATTLE DEPT OF COMPUTER SCIENCE R E CYPHER ET AL. JUL 87 1/1

UNCLASSIFIED N00014-86-K-0264 F/G 12/9 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) The Hough Transform Has O(N) Complexity on SIMD N x N Mesh Array Architectures		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) R. E. Cypher, J. L. C. Sanz and L. Snyder		8. CONTRACT OR GRANT NUMBER(s) N00014-86-K-0264
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Washington Department of Computer Science Seattle, Washington 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Information Systems Program Arlington, VA 22217		12. REPORT DATE July 1987
		13. NUMBER OF PAGES 7
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

parallel algorithms, image processing, mesh, Hough transform

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper reports on new algorithms for computing the Hough transform on mesh array architectures. The mesh is fine-grained, consisting of an N x N array of processors, each holding a single pixel of the image. The mesh array operates in an SIMD mode. Several algorithms, differing in the techniques they use, their asymptotic complexity, or the architectural resources required, are presented for computing the Hough transform. The main algorithm computes any P angles of the Hough transform in O(N + P) time and uses only a constant amount of memory per processor. All the algorithms apply to the more general

problem of computing the Radon transform of gray-level images, [illegible]

The Hough Transform Has O(N) Complexity On SIMD $N \times N$ Mesh Array Architectures

R. E. Cypher(*), J. L. C. Sanz (**) and L. Snyder (*)
(*) Department of Computer Science, University of Washington, Seattle, WA 98195
(**) Computer Science Department, IBM Almaden Research Center, San Jose, CA 95120

Abstract

This paper reports on new algorithms for computing the Hough transform on mesh array architectures. The mesh is fine-grained, consisting of an $N \times N$ array of processors, each holding a single pixel of the image. The mesh array operates in an SIMD mode. Several algorithms, differing in the techniques they use, their asymptotic complexity, or the architectural resources required, are presented for computing the Hough transform. The main algorithm computes any P angles of the Hough transform in $O(N + P)$ time and uses only a constant amount of memory per processor. All the algorithms apply to the more general problem of computing the Radon transform of gray-level images.

Introduction

Parallel computing for image processing and computer vision has recently received considerable attention [1-4]. Technological advances have made possible fine-grained architectures [5-7]. Mesh connected computers are of particular interest to the image community [9-15]. Many practical algorithms have been implemented on both fine-grained arrays [16-17], and coarse-grained meshes [18]. Theoretical advances for mesh computer algorithms have also been reported [19-20], although it will take some time before these ideas can be put into practice. This paper addresses the problem of calculating the Hough Transform, and more generally the Radon Transform, on a mesh array computer. The fastest published algorithm for this problem calculates P projections of the Hough Transform in $O(PN)$ time [8]. Algorithms will be presented here that calculate P such projections in $O(P + N)$ time.

Architectural Preliminaries

A mesh array computer consists of an $N \times N$ array of processing elements (PEs) where each PE consists of a processor and an associated memory. The PEs operate in a Single Instruction Stream, Multiple Data Stream (SIMD) mode, with all control signals coming from a single control unit. The control unit reads instructions from its private memory, decodes them, and broadcasts the control signals to the PE array. In addition to broadcasting the control information to the processors, the control unit typically sends addresses to the memory units, so every PE accesses the same memory location at a given time. Although there is no direct data connection from the controller to the array, there are situations when a number must be broadcast from the controller to the PEs. This is accomplished by using the control lines to cause the PEs "to calculate" the number one bit at a time.

There is assumed to be a special register in each PE, called a mask register, which when an instruction is sent from the controller, only those PEs with a 1 in their mask register perform the instruction; all others do nothing. This allows operations to be performed on a subset of the PEs in a data dependent manner. Of course, there are some instructions that operate on all PEs regardless of the setting of the mask registers, thus allowing the disabled PEs to be used again.

Two architectures will be used here.

The first architecture studied is a square array[1] of $N \times N$ PEs. With the top and bottom rows connected and the leftmost and rightmost columns are connected, so the interconnections logically from a torus. The input is an $N \times N$ image. This will be refereed to as the "plain array" architecture. The PEs operate in a SIMD mode under the direction of a single controller for the entire array. Each processor has its own local RAM, and unless stated otherwise, it consists of a constant number of words of memory, where each word has $O(\log N)$ bits. Processors communicate directly with their 4 closest neighbors one neighbor at a time. It is assumed that each processor has a mash register.

The second architecture is identical to the first except that in addition to the square array of PEs, it includes a tree of PEs per row of the array. The PE trees are built on top of the array, so that the row PEs form the leaf nodes of a tree. The root of each tree has an output port for the removal of results from the tree. A second controller is provided so that the PEs internal to the tree can perform one operation while the PEs in the array perform another. In this way, additional *parallelism is achieved. This architecture will be referred to* as the "augmented array" architecture. For the algorithms presented in the next section, it is sufficient that the PEs in the nonleaf levels of the trees be adders. These trees can be shown to be useful in solving a number of important vision problems [21].

[1]Notice that arrays of processors and pixels are indexed such that (0,0) is in the *lower* left corner.

The time analysis of the algorithms will measure simple operations on 1 word quantities, where each word has O(log N) bits. Such a word size is sufficient to represent values the quantities computed. Although operations on words that are longer than 1 bit are not directly supported by the bit-serial machines currently in existence, they can be implemented in software as a sequence of bit-serial operations. The time analysis will be for the worst case data.

Hough and Radon Transforms

The Radon Transform of a gray-level image is a set of projections of the image taken from different angles. Specifically, the image is integrated along line contours defined by the equation:

$$\{(x,y) \mid x\cos(\theta) + y\sin(\theta) = \rho\}$$

where θ and ρ are the orientation and the offset of the line, respectively. It may be assumed that $0 < \theta \leq \pi$, because projections are the same for θ and for $\theta + \pi$ regardless of the value of θ. The original definition of the Radon transform involves a line integral, but for digitized pictures, this integral is replaced by a weighted summation. It will be assumed that each line contour is approximated by a 1 pixel wide band. All pixels centered in a given band contribute to the value of that band. Because each band is 1 pixel wide, there are at most $\sqrt{2}$ N values of ρ for each value of θ. The parameter P will be used to denote the number of projections (values of θ) to be calculated. The computational aspects of the discrete form of the Radon transform and its implementation in pipeline architectures have been considered [22].

The Hough transform, often used to locate the edges of objects in an image, is simply the case of the Radon transform where the input image is binary. It has been implemented on different processors such as the GAPP [8], systolic arrays [23], and other pipeline architectures [24]. Special architectures for computing peaks of the Hough transform have been proposed [25] and built [26]. In addition, Kushner *et al.* [27] studied the problem of implementing the Hough Transform on the MPP and concluded that because the projections are at various orientations, the problem "is of a form that the fixed geometry of the MPP cannot easily handle." There follows five new algorithms for the Radon (Hough) Transform on mesh arrays having different time complexities and resource requirements.

One way to calculate either Transform on a plain array is to rotate the columns of the image until all of the pixels that lie in the same band (line contour) for a given projection angle are in the same row. The projection is then calculated by totaling the values for each band by using horizontal shifts and adds. The first four algorithms are based on this general approach. The first algorithm presented here is faster by a constant factor than the best publish algorithm [8] and the remaining algorithms are asymptotically faster.

Algorithm 1

The first algorithm operates on a plain array with a constant number of words of memory per PE. It requires O(PN) time to calculate P projections of the Transform. The case where $\frac{\pi}{2} < \theta \leq \frac{3\pi}{4}$ will be examined first. The values of θ are treated in order starting with $\frac{\pi}{2}$ and increasing through $\frac{3\pi}{4}$. For each value of θ, the controller first broadcasts the values $\sin(\theta)$ and $\cos(\theta)$ to all PEs. Each PE calculates

$$d = x\cos(\theta) + y\sin(\theta),$$

where x and y are the row and column coordinates of the pixel in the PE. The value d is the distance from the origin to the line passing through the point (x, y) and perpendicular to θ. Next each PE calculates

$$\rho = \text{floor}(d)$$

the distance from the origin to the 1 pixel wide band passing through (x,y) at the desired angle. Then each PE calculates

$$v = \text{ceiling}(\rho/\sin(\theta))$$

the smallest value on the y-axis within the band containing the point (x,y). Each PE then creates a record containing its pixel value and the variables x, y and v. These records are shifted downward (cyclically, because of the connections between the top and bottom rows) until each record is in row r where r = v mod N. (See Figure 2.) For instance, if the record originally in PE (3, 10) has a v value of 1, the record is shifted down 2 rows. After this downward shifting, each row r will have the pixel values of all points in bands that cross the y-axis at distance v from the origin where v = r mod N.

Because $\frac{\pi}{2} < \theta \leq \frac{3\pi}{4}$, there are at most two bands with different values of v on the same row r. Also at most two pixels in the same column of the image lie within the same band because each band is 1 pixel wide. To avoid shifting one record on top of another, the downward shifts should be performed in two stages, one for pixels with even y coordinates and one for odd y coordinates. Once the records have been shifted downward to the proper row r = v mod N, they are stored for future use in calculating the next projection. Then, the pixel values for records with odd y coordinates are added to those with even y coordinates. Next, the total of each band is calculated by shifting the pixels left one column and adding, then shifting them left two columns and adding, then shifting them left four columns and adding, etc. Because each row may have the pixels for two bands within it, this summation of pixels across the rows must be performed in two passes, one for each band. The totals accumulated in the first column form the desired projection. The next projection is calculated in the same manner, starting with the records in the PEs where they were stored after performing the vertical shifts in the calculation of the previous projection. By performing the projections in order and storing the records after down shifting each projection, the later projections can take advantage of the shifts performed for the earlier projections. As a result, the total number of downward shifts is reduced. In fact, it can be shown that only O(P + N) downward shifts are required for all of the projection angles in the range $\frac{\pi}{2} < \theta \leq \frac{3\pi}{4}$. The projections for $\frac{\pi}{4} < \theta \leq \frac{\pi}{2}$ are calculated in the same way starting with the original unshifted image and processing the θ values in *decreasing* order.

There is a problem with using this same algorithm to calculate the projections for $0 < \theta \leq \frac{\pi}{4}$ and $\frac{3\pi}{4} < \theta \leq \pi$. These projections use bands that are more vertical than horizontal, so many bands with different values of v will have the same value of r. Because these bands would have to be totaled sequentially, the total time for the algorithm would increase. One solution is to modify the algorithm so that role of rows and columns is reversed. Then each band would be shifted sideways (instead of down) until each occupies a single column of PEs; totaling the bands would be performed within columns. This technique would guarantee that no more than 2 bands would occupy the same column just as no more than 2 bands could occupy the same row when $\frac{\pi}{4} < \theta \leq \frac{3\pi}{4}$.

A different technique is presented here, since it can be easily modified to create an algorithm for an augmented array. This algorithm calculates the transpose of the image before calculating the projections for $0 < \theta \leq \frac{\pi}{4}$ or $\frac{3\pi}{4} < \theta \leq \pi$. To obtain the transpose, the original image is shifted in 3 stages: In the first stage, the pixel in PE (x,y) is shifted down x times. In the second stage, the pixel in P(x,y) after the first stage is shifted to the right y times. In the third stage, the pixel in PE (x,y) after the second stage is shifted down x + 1 times. The result is actually the reflection of the transpose about a vertical axis, but this is sufficient to guarantee that no more than 2 bands are ever sent to the same row.

Algorithm 2

In the Transform algorithm presented above, there are O(P + N) downward shifts and O(PN) horizontal shifts. The second algorithm is identical except that it uses the augmented array to perform the horizontal shifts and adds. The augmented array can calculate totals for all rows in parallel in O(log N) time. More importantly, the row totals for successive projections can be pipelined, so that P projections can be calculated in O(P+N) time. As soon as the totals for one projection angle start up the PE trees, the vertical shifts for the next projection angle are started. This overlap of horizontal totaling and vertical shifting is possible because there are two controllers present in the augmented array. Again, it is important that the (reflection about a vertical axis of the) transpose is used so that all totalling occurs along rows and can thus use the trees of PEs. As in the first algorithm, the second algorithm uses only a constant number of words of memory per PE.

Algorithm 3

The third algorithm can perform the Hough Transform on a certain type of plain array machine in O(P+N) time. The required plain array machine must have two features not needed for the O(PN) time algorithm. First, each PE must have O(P) words of memory. Second, the PEs must have the ability to perform *independent* addressing, that is, rather than all PE's referencing the same data address, they can reference different addresses. Independent addressing can be implemented in SIMD execution mode using indirection. The algorithm uses the same techniques of performing vertical shifts to place the pixels from each band into the same row of PEs, and horizontal shifts to total the pixels in each band. Whereas the first algorithm alternated between performing the vertical shifts and totaling the bands and the second algorithm overlaps the totaling for the bands of one projection with the vertical shifts for the next projection, the third algorithm performs the vertical shifts for all of the projections before calculating any of the totals for the bands. First the vertical shifts for the projections in the range $\frac{\pi}{4} < \theta \leq \frac{3\pi}{4}$ are performed in order; then the vertical shifts for the projections $\frac{\pi}{4} < \theta \frac{\pi}{2}$ are performed. Next, the transpose of the image (reflected about the vertical axis) is calculated. Then the vertical shifts for $0 < \theta \leq \frac{\pi}{4}$ and for $\frac{3\pi}{4} < \theta \leq \pi$ are performed. Each PE uses a variable called the "buffer array" to hold the 2^P entries. After the vertical shifts for the i-th projection angle are performed, the shifted pixels are stored in the buffer array either in location i or in location i + P. Specifically, the pixels that have nonnegative v values (refer to algorithm 1 for the definition of v values) are stored in location i and the pixels with negative v values are stored in location i + P. The buffer array is initialized to all 0s before the pixels are stored in it, so any location not containing a pixel contains a 0. The total time for the algorithm to this point is O(P+N).

Each PE contains a buffer array with 2^P entries. The entries corresponding to a single band of a projection all lie in the same row of PEs and they are all at the same offset within the buffer arrays. The totals for the bands are now calculated in O(P+N) time. Each PE has a variable, called band_total, that is initialized to 0. To calculate the band totals, it begins by adding the contents of buffer location i to the band_total, where i is the column number of the PE holding the buffer. (It is this operation that necessitates the independent addressing.) Next, it shifts the band_totals one column to the right, and adds the contents of buffer location (i-1) mod N to the band_totals. Again the band_totals are shifted one column to the right, and added to the contents of buffer location (i-2) mod N. This procedure of shifting and adding is repeated N times. The effect of these operations is shown in Figure 3. If 2P < N, then some PEs will attempt to access buffer locations that do not exist; such PEs are disabled while the buffer is being accessed. Once the N horizontal shifts and adds have been performed, each band_total variable contains the total for a different band. Assuming that the leftmost column of PEs is connected to the array's I/O ports, the band_total variables can be shifted to the left N times to remove the result from the array. This totaling procedure uses O(N) time. If 2P ≤ N, the Transform is complete. If 2P > N, then set the band_total variables to 0 again and repeat the above procedure an additional ceiling (2P/N)-1 times. In this case, the totaling requires O(P) time. Either way, the entire algorithm requires O(P+N) time.

Algorithm 4

The fourth algorithm uses a plain array with no independent addressing and O(P) words of memory per PE to calculate P projections of the Transform in O(P log P + N) time, assuming that $2P \leq N$. Again, each PE has a buffer array with 2P entries, initialized to 0. The technique as was used in the third algorithm is used to store the vertically shifted pixels into the buffer arrays in O(P + N) time. The fourth algorithm differs, however, in the way the bands are totaled. In the third algorithm, independent addressing was used so that the N band_total variables in each row accessed different locations in the buffer arrays. In the fourth algorithm, with no independent addressing available, the contents of each buffer array is rotated downward i mod (2P) positions, where i is the number of the column of the PE holding the array. These downward rotations are cyclic, so that the contents of buffer array location j is moved to buffer array location (i + j) mod (2P). The technique to accomplish these downward shifts will be explained shortly. Once the downward rotations are completed, the band_total variables are used to calculate the totals for the bands: First they are initialized to the contents of location 1 in the buffer arrays. Then they are shifted one more column to the right and added to the contents of location 2 in the buffer arrays. This shifting and adding procedure is repeated until each band_total is the sum of N entries. (This band totaling procedure is illustrated in Figure 4.) The results are then shifted out of the array with N left shifts. Because it was assumed that $2P \leq N$, this is the desired result. The horizontal shifts and additions require O(N) time.

One way to perform the downward rotations of the buffer arrays would be to process in parallel only those columns that are being rotated the same amount: all other columns would be disabled. The contents of each buffer array in the enabled columns would be copied to a temporary array in the same PE. Then the contents of this temporary array would be copied to the buffer array in the rotated position. To rotate by a single amount requires O(P) time. Because there are O(P) different amounts to be rotated, the entire rotation procedure would require $O(P^2)$ time.

A better technique is based on the idea of a barrel shifter: it performs the rotation in O(P log P) time. Only shifts that are powers of 2 are performed. First, some buffer arrays are shifted downward 1 position, then some are shifted downward 2 positions, then some are shifted downward 4 positions, etc. The PEs that are enabled during a given shift are chosen according to the following rule: Let i be the column number of the PE, and let k = i mod (2P) be the number of positions that its buffer array will be shifted. Then when a downward shift of 2^j positions is being performed, only those PEs are enabled which have the j-th bit of k equal to 1, assuming that the least significant bit is the 0th bit. A total of ceiling (log 2P) different shifts are performed, and each shift requires O(P) time, so all of the downward shifts can be performed in O(P log P) time.

In the above analysis, it was assumed that $2P \leq N$, and so a total of O(P log P + N) time is required. If $2P > N$, then the entire procedure given above can be used to calculate the first N/2 projections in O(N log N) time, the next N/2 projections in an additional O(N log N) time, etc. A total of ceiling (2P/N) iterations are required, yielding a total time of O(P log N).

Algorithm 5

Finally, the fifth algorithm uses a plain array, no independent addressing and only a constant number of words of memory per PE to calculate P projections of a Transform in O(P+N) time. The technique used differs significantly from the techniques used in the four previous algorithms. The previous algorithms all performed vertical shifts on the image in order to simplify the totaling of the pixels in each band. This algorithm, however, does not perform such vertical shifts. Instead, the total for a band is calculated by having a variable, called "band_total", visit all of the pixels in its assigned band. As a band_total encounters a pixel in its band, it adds the pixel's gray level to itself. An informal description of the algorithm is given next.

To understand the algorithm, it is useful to examine how a single band_total variable is moved across the image. This band_total is assigned a particular value of "v" and "θ" (refer to algorithm 1 for a definition of v). It must move across the image in such a manner that it visits all of the pixels (x,y) where

$$\text{floor}\,(x\cos(\theta) + y\sin(\theta)) = v.$$

The set of pixels that must be visited by this band_total will be referred to as the pixels "owned" by the band_total. An example of the set of pixels owned by a band_total is shown in Figure 5.

In the following it will be assumed that $\frac{\pi}{2} < \theta \leq \frac{3\pi}{4}$, though the same techniques apply to the other cases. In particular, when $\frac{\pi}{4} < \theta \leq \frac{\pi}{2}$, the roles of up and down shifts are reversed. The remaining cases are identical except that the roles of the rows and columns are reversed.

Because $\frac{\pi}{2} < \theta \leq \frac{3\pi}{4}$, each band_total owns at most 2 pixels in each column. One way the band_total could visit these pixels is to visit each PE that contains them. This has the disadvantage that the band_total must shift vertically to access both of the pixels it owns in a single column. The algorithm presented here avoids these vertical shifts by performing a single downward shift of the image. There are thus two versions of the image in each PE, the original version and the down-shifted version. The band_total is moved across the image, visiting the PE containing the lower of the band_total's pixels in each column.

The set of PEs visited by the band_total in Figure 5 is shown in Figure 6. The set of PEs that a band_total must visit will be referred to as the PEs that are "owned" by the band_total. In order for a band_total to visit all the PEs it owns, it is placed initially in the leftmost PE. It is then shifted one column to the right and then optionally shifted up one row to visit the next PE that it owns. This process of shifting to the right and optionally shifting up is repeated until the band_total has visited all of its PEs. The decision as to whether or not the band_total should be shifted upwards is based on a local calculation that will be explained below.

Having examined the behavior of a single band_total, it is now possible to explain how the band_totals operate in parallel. Consider the calculation of a single projection angle θ, still assuming that $\frac{\pi}{2} < \theta \leq \frac{3\pi}{4}$. This projection is calculated by placing the band_totals in the first column of PEs, each in the PE that it owns. The PE owned by a given band_total sets its column_contrib (column contribution) variable to the sum of the gray levels of the pixels owned by that band_total in the column. The band_total adds this column_contrib to its current value. The band_totals are then shifted right to the second column and some are shifted up so that all are placed in the PE that they own in the second column. Again, the PEs set their column_contrib variables and these variables are added to the band_totals. This procedure is repeated until all columns have been visited. At any one time, the band_totals for the given projection angle θ are all located in the same column of PEs. Note that collisions between band_totals are impossible because the paths of two band_totals with the same value of θ can never cross.

To calculate the projection for a given angle theta, it is necessary to visit the PEs in the first column, the PEs in the second column, etc., until all of the columns have been visited. Thus, the PEs that are used in calculating one projection of the Transform form a wave propagating from the left to the right. Accordingly, it is possible to pipeline the calculation of the Transform by starting the calculation for the second projection angle as soon as the first has finished using the PEs in the first column. As many as N projections can be pipelined in this manner at one time. Because the calculation of a single projection requires O(N) time, and because the calculation of P projections can be pipelined, the calculation of the entire Transform can be performed in O(N + P) time.

The above discussion ignored a number of details of the algorithm. In particular, it did not specify how the band_totals determine when to shift up, how the column_contrib variables are calculated, or how bands that do not include pixels in both the first and last columns are handled. These issues are addressed next.

When the calculation of a new projection angle θ is started, the values $\cos(\theta)$ and $\sin(\theta)$ are broadcast from the controller to the PEs in the first column. These values are shifted right whenever the corresponding band_totals are shifted right. In addition, each band_total is accompanied by a band_number identifying which band it is following. After performing a right shift of the band_total, band_number, $\sin(\theta)$ and $\cos(\theta)$, the PEs calculate

$$d = x\cos(\theta) + y\sin(\theta) \text{ and } \rho = \text{floor}(d).$$

Then each PE determines if it is the lowest PE in its column with its value of ρ. This is accomplished by shifting the ρ values up one row. Each PE compares the received ρ value with its own ρ value, and if they match, sets its own ρ value to infinity. At this point the PEs with finite ρ values are the ones owned by some band_total for the current projection angle θ. Next, the (possibly infinite) ρ values are shifted down one row. The down-shifted ρ values determine whether or not there are two pixels in the given column that lie in the same band: If a PE's own ρ value is finite and the ρ value that is received from the PE above it is finite, then the PE knows that it contains the only pixel in its column lying in its band. From this information each PE with a finite ρ value sets its column_contrib variable to the sum of the gray levels of the pixels in its column and band. Next, each band_total examines the ρ value of the PE in which it is located. If the PE has a finite ρ value matching the band_number, then the band_total is in the correct PE. Otherwise, the band_total and band_number are shifted up one row. The column_contrib is then added to the band_total.

So far, band_totals have only been created for bands containing pixels in the leftmost column. However, some bands may not contain any pixels in the leftmost column. Such a band is handled in a similar manner, with the only difference being that its band_total variable is created in the leftmost column having a pixel in the band. Also, it has been assumed that all of the bands continue all the way to the rightmost column, although in reality some do not (they go off the top or bottom of the image). When a band_total goes off the top of the image before reaching the right hand edge, it arrives in the bottom row of PEs because of the toroidal connections between the top and bottom rows. It continues to follow the wave of processing associated with its projection angle θ, but it is no longer increased by the column_contrib variables it encounters. The description given above was a simplification because it assumed that each PE has a single band_total variable at a given time. In reality, a PE can have two band_total variables at once: one that is actively visiting PEs in its band and one that has completed its calculations and has gone off the top or bottom of the image.

The algorithm detailed more fully in a technical report [28] requires 20P + 48N + 4 communication operations, and a similar number of local operations, to calculate P projections. While this is fairly fast, it is possible to improve the speed in certain situations. If the same set of projection angles is used repeatedly for calculating the Transform for many images, the values of $\sin(\theta)$ and $\cos(\theta)$ need not be shifted across the image, since the path of each band_total variable across the image is completely determined by the values of θ. Hence, the paths can be pre-computed and stored in the PEs. Each PE can contain a single bit for each time a "where" clause in the program is executed. This bit indicates whether or not the "where" clause is true at the given time. This requires an additional P bytes of memory per PE. When this approach is used, only 6P + 14N + 4 communication operations are required to calculate P projections. In addition, this approach requires fewer local operations and no multiplications. Finally, it should be noted that in any case the multiplications can be avoided if the values of $\cos(\theta)$ and $\sin(\theta)$ are sufficiently accurate, since the function $x \cos(\theta) + y \sin(\theta)$ can then be calculated by using forward differencing.

Conclusion

Five new algorithms for calculating Radon (Hough) Transform on mesh arrays have been presented. The asymptotically fastest algorithm previously published requires $O(PN)$ time to calculate P projections of the Transform, while three of the algorithms presented here require only $O(P + N)$ time. In addition to the theoretic value of these algorithms, the authors hope that they will prove fast enough to have practical applications. Typically values of the parameters N and P are 512 and 180, respectively, so the algorithms presented here do seem to offer a time savings for realistic problems.

Acknowledgement

The work of R. Cypher was supported by the National Science Foundation graduate fellowship program. The work of L. Snyder was supported in part by NSF grant DCR8416878.

References

[1] A. P. Reeves, "SURVEY: Parallel Computer Architectures for Image Processing", *Computer Vision, Graphics, and Image Processing* 25, pp. 68-88, 1984.

[2] T. J. Fountain, "Array Architectures for Iconic and Symbolic Image Processing", *Proceedings of the 8th International Conference on Pattern Recognition*, pp. 24-33, 1986.

[3] V. Cantoni, S. Levialdi, *Pyramidal Systems for Computer Vision*, NATO ASI Series, Computer and Systems Science, vol. 25, Springer Verlag, 1987.

[4] M. J. B. Duff, ed., *Intermediate-Level Image Processing*, Academic Press, London, 1986.

[5] S. Tanimoto, T. Ligocki, R. Ling, "A Prototype Pyramid Machine for Hierarchical Cellular Logic", in Uhr, L. (ed.), *Parallel Hierarchical Computer Vision*, Academic Press, Orlando, FL, to appear in 1987.

[6] D. Hillis, *The Connection Machine*, MIT Press, 1985.

[7] K. E. Batcher, "Design of a massively Parallel Processor", *IEEE Transactions on Computers*, C-29(9):826-840, 1980.

[8] T. Silberg, "The Hough Transform in the Geometric Arithmetic Parallel Processor", IEEE Workshop on Computer Architecture and Image Database Management, pp. 387-391, 1985.

[9] S. Levialdi, "On Shrinking Binary Picture Patterns", *Communications of the ACM*, 15(1):7-10, 1972.

[10] A. Rosenfeld, "Parallel Image Processing Using Cellular Arrays", *IEEE Computer*, pp. 14-20, 1983.

[11] C. R. Dyer, A. Rosenfeld, "Parallel Image Processing by Memory-Augmented Cellular Automata", *IEEE Transactions on PAMI*, PAMI-3:29-41, 1981.

[12] M. J. B. Duff, "Review of the CLIP Image Processing System", National Computer Conference, Anaheim, California 1978.

[13] S. Wilson, "The Pixie-5000: A Systolic Array Processor", *Proceedings of the IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Miami, Florida, November 18-20, 1985.

[14] NCR Microelectronics Division, Product Description ncr45cg72, NCR Corporation, Dayton, Ohio, 1984.

[15] T. J. Fountain, "Plans for the CLIP7 Chip", *Integrated Technology for Parallel Image Processing*, in S. Levialdi (ed.), Academic Press, pp. 199-214, 1985.

[16] J. L. Potter, "Image Processing on the Massively Parallel Processor," *IEEE Computer*, pp. 62-67, 1983.

[17] M. J. B. Duff, "Real Applications on CLIP4", *Integrated Technology for Parallel Image Processing*, in S. Levialdi (ed.), Academic Press, 1985.

[18] K. N. Matthews, "The CLIP7 Image Analyser - a Multi-Bit Processor Array", Ph.D. Thesis, University of London, 1986.

[19] D. Nassimi, S. Sahni, "Finding Connected Components and Connected Ones on a Mesh-Connected Parallel Computer", *SIAM J. Computing*, 9(4):744-757, 1980.

[20] R. Miller, Q. Stout, "Geometric Algorithms for Digitized Pictures on a Mesh-Connected computer", *IEEE Transactions on PAMI*, PAMI-7(3):216-228, 1985.

[21] R. E. Cypher, J. L. C. Sanz, *Parallel Algorithms and Architectures for Image Processing and Computer Vision*, in preparation.

[22] J. L. C. Sanz, E. B. Hinkle, "Computing Projections of Digital Images in Image Processing Pipeline Architectures", IEEE Transactions on ASSP, vol. ASSP-35, pp. 198-207, February 1987.

[23] H. Chuang, C. Li, "A Systolic Array for Straight Line Detection by Modified Hough Transform", IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management, pp. 300-304, 1985.

[24] J. L. C. Sanz, I. Dinstein, "Projection-based Geometrical Feature Computation for Computer Vision: Algorithms in Pipeline Architectures", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, January 1987

[25] H. Li, M. Lavin, R. LeMaster, "Fast Hough Transform: A Hierarchical Approach", CGVIP 36:139-161, January 1987.

[26] J. Dituri, F. M. Rhodes, D. Dudgeon, "Hough Transform System", 1986 Workshop on VLSI Signal Processing, November, Los Angeles, California.

[27] T. Kushner, A. Wu, A. Rosenfeld, "Image Processing on the MPP", *Pattern Recognition*, 15:121-130, 1982.

[28] R. E. Cypher, J. L. C. Sanz, L. Snyder, "The Hough Transform Has O(N) Complexity on SIMD N × N Mesh Array Architectures", University of Washington, Technical Report #87-07-01, July 1987

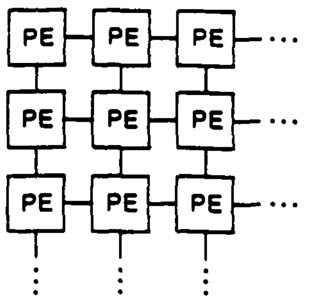


Figure 1. Mesh connected array.

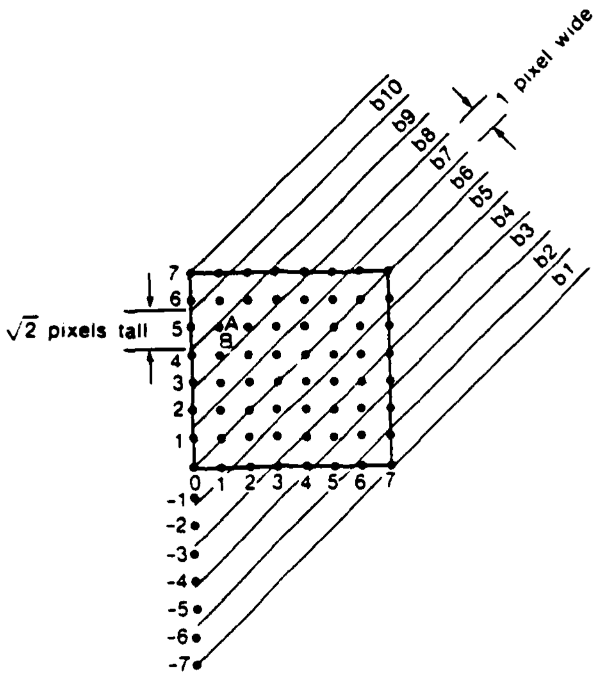
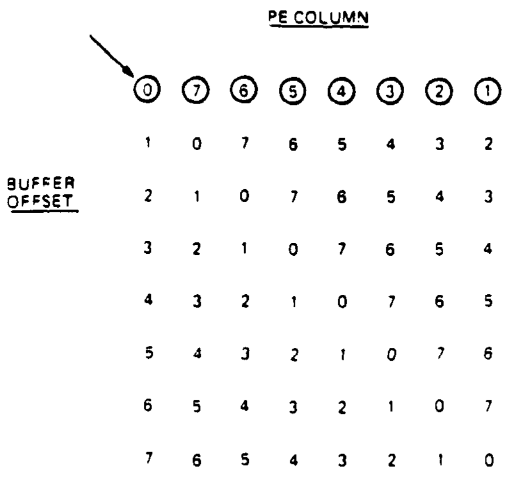


Figure 2. Bands over an 8 × 8 image.

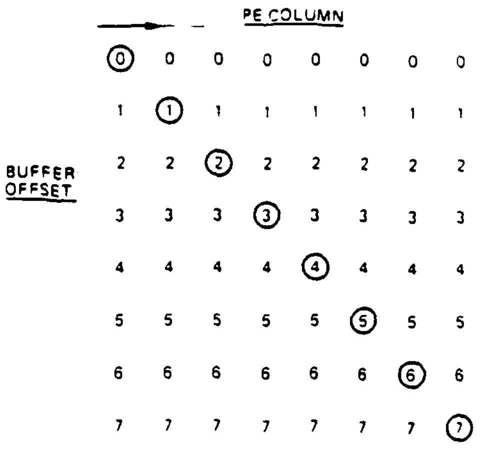


PE COLUMN

BUFFER OFFSET

⓪	⑦	⑥	⑤	④	③	②	①
1	0	7	6	5	4	3	2
2	1	0	7	6	5	4	3
3	2	1	0	7	6	5	4
4	3	2	1	0	7	6	5
5	4	3	2	1	0	7	6
6	5	4	3	2	1	0	7
7	6	5	4	3	2	1	0

Figure 3. Numbers indicate band to which pixel belongs.
Circled numbers are accessed first.
The arrow indicates the direction in which each
band_total variable moves.



PE COLUMN

BUFFER OFFSET

⓪	0	0	0	0	0	0	0
1	①	1	1	1	1	1	1
2	2	②	2	2	2	2	2
3	3	3	③	3	3	3	3
4	4	4	4	④	4	4	4
5	5	5	5	5	⑤	5	5
6	6	6	6	6	6	⑥	6
7	7	7	7	7	7	7	⑦

Figure 4. Same as in Figure 3.

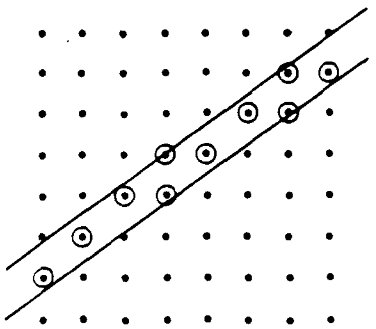


Figure 5. Pixels owned by a given band.

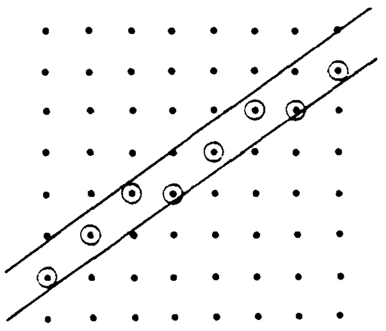


Figure 6. PE's owned by a given band.